Using Arithmetic Transform for Verification of Datapath Circuits via Error Modeling

Katarzyna Radecka and Zeljko Zilic
McGill University, Dept. of Electrical and Computer Engineering,
3480 University St., Montréal, Québec H3A 2A7, Canada
{kasiar,zeljko}macs.ece.mcgill.ca

Abstract

*In this paper, we consider verification under error-model assumption. We exploit the algebraic properties of the arithmetic transforms that are used in compact graph-based representations of arithmetic circuits, such as *BMDs. Verification time can be shortened under assumption of corrupting a bounded number of transform coefficients. Bounds are derived for a number of test vectors, and the vectors successfully verified arithmetic circuits under a class of error models derived from recently proposed basic design error classes, including single stuck-at faults.*

1. Introduction

Modern microprocessors, embedded and signal processors as well as communication ASICs utilize various arithmetic circuits in their datapaths. These arithmetic circuits vary in their area, delay and power constraints. Hence, many different realizations of arithmetic circuits can be found, from custom to those that are modified from the standard library elements. Their design, testing and verification poses a major challenge.

Verification of arithmetic circuits has exposed limits of methods based on Decision Diagrams (DDs). The original Reduced Ordered Binary Decision Diagrams (ROBDDs) present a canonical reduced representation of a truth table. They could be used to verify in polynomial time circuits such as adders. However, ROBDDs are of exponential size [1] even for multipliers. Numerous extensions to the ROBDDs were proposed that made the verification of arithmetic circuits more efficient. The most relevant are the extensions to the world-level diagrams, such as *BMDs [3] or EVBDDs [6], for multi-output Boolean functions. All such diagrams are included in World Level Decision Diagrams (WLDDs). The common limitation of any WLDDs is its inability to represent dividers [9] and the more complex datapath operators by polynomial size diagrams.

While WLDDs enabled the verification of circuits such as multipliers, we believe that their current use has serious shortcomings. First, they have been used only for equivalence/model checking, that compares a function implementation against its specification. This approach is analogous to comparing two abstraction levels for each input combination. However, under some conditions, a smaller number of comparisons could be performed. For example, if a set of possible errors were to be known, one could devise a set of *test vectors* that can detect all such faults. Further, it would be useful if the verification vectors could be applied for the testing purposes, i.e. for a single stuck-at fault error model.

In this paper, we explore the theoretical bounds, and present the experimental demonstration of such a verification scenario. We exploit the properties of the Arithmetic Transform, which presents the underlying mechanism behind the word-level DDs. We present the basic test vector generation scheme and some of its optimizations, together with deriving an upper bound on the number of test vectors under the assumption of bounded error in spectral domain. Finally, we demonstrate that the same test vector can be applied successfully to a model of design errors [2], considered recently for verification by error modeling.

2. Arithmetic Transform

The Arithmetic Transform (AT), also known as integer-valued Reed-Muller (RM) polynomials [5] extend the traditional RM forms by allowing the integer function values, while the inputs remain Boolean. The RM forms are obtained by employing a Davio expansion around each input variable $x, y, \ldots$ as follows:

$$f = f\,|_{x=0} + x\,(f\,|_{x=1} - f\,|_{x=0}) \qquad (1)$$

The arithmetics is performed modulo 2; consequently, "+" and "-" denote an XOR operation. The AT is obtained by using in Equation 1 the integer addition instead.

0-7695-0613-5/00 $10.00 © 2000 I

Integer Encoding	Number Norm $\|x\|$
Unsigned	$\sum_{i=0}^{n-1} x_i 2^i$
Sign extended	$(1-2x_{n-1})\sum_{i=0}^{n-2} x_i 2^i$
1's complement	$\sum_{i=0}^{n-2} x_i 2^i - x_{n-1}(2^{n-1}-1)$
2's complement	$\sum_{i=0}^{n-2} x_i 2^i - x_{n-1}2^{n-1}$

Table 1. Norms of Integer Encodings

The Arithmetic Transform of a multi-output Boolean functions is calculated by applying the expansion from Equation 1. This expansion leads to a polynomial:

$$f = \sum_{i_1=0}^{1}\sum_{i_2=0}^{1}\cdots\sum_{i_n=0}^{1} c_{i_0 i_1 \ldots i_{n-1}} x_0^{i_0} x_1^{i_1} \ldots x_{n-1}^{i_{n-1}} \qquad (2)$$

Integer coefficients $c_{i_1 i_2 \ldots i_n}$ are called the *arithmetic spectrum*. Each coefficient multiplies a product $x_0^{i_0} x_1^{i_1} \ldots x_{n-1}^{i_{n-1}}$, where each Boolean variable x_j is raised to an exponent i_j. When an exponent i_j is equal to 1, the variable appears in the product term, otherwise it does not. Hence, each coefficient multiplies a product term consisting of a subset of all variables. There are 2^n such subsets, and the arithmetic spectrum coefficients can be associated with these subsets, or equivalently with the subset characteristic functions.

To quickly derive arithmetic spectra for datapath circuits, we use an auxiliary norm function $|x|$, equal to the integer value that a binary-represented number takes. For an unsigned n-bit variable $x = x_0 x_1 \ldots x_{n-1}$, this value is calculated as $|x| = \sum_{i=0}^{n-1} x_i 2^i$. Table 1 contains norms of frequently used integer data type encodings.

To obtain the Arithmetic Transform of the adder, we consider the numerical value of the sum of two n-bit unsigned numbers x and y, calculated as:

$$|x+y| = \sum_{i=0}^{n-1}(x_i + y_i)2^i.$$

Comparing with Equation 2, we notice that this is a polynomial with integer coefficients representing multiple-output function of arguments x_i, y_i where $i = 0, \ldots, n-1$. In conclusion, the AT of the addition operation has $2n$ nonzero spectral coefficients.

In the similar way, the subtraction operation is obtained by replacing the arithmetic "+" with a "-" sign. For example, for sign-extended encoding, the difference can be obtained as:

$$x - y = \sum_{i=0}^{n-2}(x_i - y_i)2^i - 2x_{n-1}\sum_{i=0}^{n-2} x_i 2^i + 2y_{n-1}\sum_{i=0}^{n-2} y_i 2^i.$$

Multipliers can be represented in a straightforward way by using $O(n^2)$ spectral coefficients:

$$|x * y| = \sum_{i=0}^{n-1} x_i 2^i \times \sum_{i=0}^{n-1} y_i 2^i$$

resulting in n^2 spectral coefficients after the sums are multiplied out. In practice, this number can be reduced to $2n$ by keeping the polynomial in the above factored form.

The cases of multipliers and adders have been addressed in a similar way in the construction of *BMDs [3] (and all subsequent DDs). The extension to the more complex arithmetic expressions can be done as follows. A multiple-output Boolean function will be represented by a single polynomial (i.e. its arithmetic spectrum). For example, a simple expression

$$|x + yz| = \sum_{i=0}^{n-1} x_i 2^i + \sum_{i=0}^{n-1} y_i 2^i \times \sum_{i=0}^{n-1} z_i 2^i$$

leads to the arithmetic spectrum for the expression $x + yz$. Any linear filter with m coefficients $a_1, a_2, \ldots a_m$ applied to n-bit integers, $x_1, x_2, \ldots x_m$ has mn spectral coefficients:

$$|a_1 x_1 + a_2 x_2 + \ldots a_m x_m| = \sum_{i=0}^{n-1}(a_1 x_{1i} + a_2 x_{2i} + \ldots + a_m x_{2m})2^i$$

The application of the AT to the alternative data types results in the spectra of the size comparable to that for the unsigned integer encoding.

2.1. Calculation of Arithmetic Spectra

The Arithmetic Transform of an arbitrary multi-output Boolean function can be obtained by multiplying the vector of function values, considered as integers, by the transform matrix. The transform matrix is defined recursively as:

$$T_n = \begin{bmatrix} T_{n-1} & 0 \\ -T_{n-1} & T_{n-1} \end{bmatrix}, \quad T_0 = 1 \qquad (3)$$

The transform matrix has 2^{2n} entries, and the transform, obtained by multiplying T_n with the vector of values, requires $O(2^{2n})$ operations.

More efficient is the Fast Arithmetic Transform, which in $O(n2^{n-1})$ time and $O(2^n)$ space recursively employs the expansion from Equation 1.

This approach is best used in conjunction with DD representations, to reduce its execution time and produce graph representations such as *BMDs.

The arithmetic spectrum can be obtained as a polynomial that is interpolated from the values that a function takes. Another way of obtaining the arithmetic spectrum is derived from this interpretation. It is most useful to consider multi-output Boolean functions $f : 2^n \mapsto 2^m$ through the Boolean lattice structure $B = 2^n$. The *partial order* relation $\prec$ is defined on Boolean vectors (points in the lattice 2^n). We say that $y \prec x$ if the 0 coordinates in x are the subset of the 0 coordinates of y. For example $0010 \prec 1010$. Incomparable

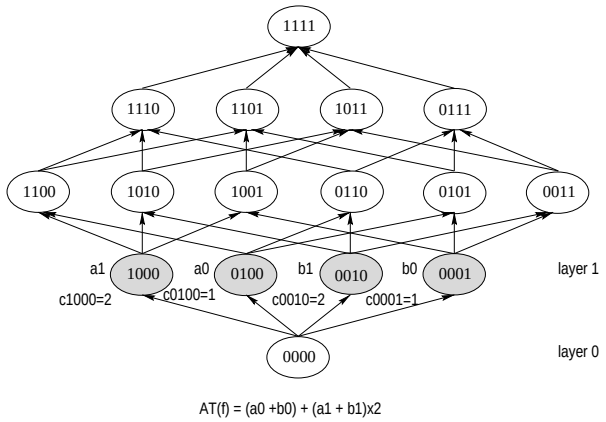

Figure 1. Lattice Structure 2^4 - Transform of Adder

vectors exist, such as 1010 and 0110. Vectors with i ones are said to belong to the same $i^{th} layer$ in the lattice. For n-variable functions, $i^{th} layer$ contains $\begin{pmatrix} n \\ i \end{pmatrix}$ vectors.

The Arithmetic Transform can be obtained by traversing the lattice in the increasing order of points. It can be shown that at each point x, the transform coefficient c_x can be calculated by subtracting all the preceding coefficients from the function value at x:

$$c_x = f_x - \sum_{y \prec x} c_y \tag{4}$$

Consider, for example, transforming the adder function $a + b$, for the 2-bit unsigned encoding: $a = a_1 a_0$ and $b = b_1 b_0$. The spectral coefficients are generated by applying Equation 4 in the lattice order, i.e. $c_{0000} = f_{0000} = 0 + 0 = 0$, $c_{0001} = f_{0001} - c_{0000} = 0 + 1 - 0 = 1$, $c_{0010} = f_{0010} - c_{0000} = 0 + 2 - 0 = 2$, $c_{0100} = 1$ and $c_{1000} = 2$. All other coefficients are 0, , as inscribed on Figure 1 where the nonzero coefficients are highlighted. For unsigned arithmetic functions, all such traversals result in forms whose nonzero coefficients are in the layer 1 (for adders) or layer 2 (multipliers) of the lattice.

In our case, the arithmetic spectrum will be used as a specification of the arithmetic operation. The shape of the polynomial for a given arithmetic operation is known and it depends only on the data type used; the addition example uses unsigned numbers. We use the knowledge of the shape of the representation polynomial to verify if the circuit matches it. We are especially interested in the minimum amount of comparisons needed under the error modeling, i.e. when all possible errors are given.

3. Error Models

Each faulty circuit can be modeled by an error superimposed on the correct circuit. Identification of accurate and robust error models is an important step in developing testing and verification methods.

In verification by error modeling, a set of test vectors is found to verify that the circuit contains no error included in the model. A circuit can be treated as a *black box*, by which the description of the design error can be obtained by subtracting the responses of the erroneous circuit from the corresponding responses of its (correct) specification. This *additive error model* is directly useful in conjunction with the Arithmetic Transform. Under the assumption of an error of bounded spectral complexity, we derive efficient verification methods.

3.1. Additive Error Model

Any error can be modeled by a quantity added to the circuit output. The operation $\tilde{f}$ of the faulty circuit is represented by the additive error model as a sum of the correct output and an error e, i.e. $\tilde{f} = f + e$. The Arithmetic Transform is linear, and satisfies the equation:

$$AT(\tilde{f}) = AT(f + e) = AT(f) + AT(e).$$

The "size" of the error will be measured in terms of the number of spectral coefficients in $AT(e)$.

For the considered verification scheme, we treat each design error as an additive error. Although the value of e for each such error can be obtained by simulations of the faulty and correct circuits, and subtracting their outputs, we emphasize that this model, and the analysis to follow, do not require the explicit identification of the error.

3.2. Arithmetic Transform of Basic Design Errors

We defined an error model through its Arithmetic Transform. The question of relating the additive error model to the models used more often in practice is addressed next. We consider the design error classes proposed in [2]. Most of these classes can be described as the additive errors with few spectral coefficients. The basic error types identified in [2] are the bus errors: Bus Order Error (BOE), Bus Source Error (BSE), Bus Driver Error (BDE), Bus Single Stuck Line (SSL) Error and Module Substitution Error (MSE) When the buses are considered in isolation, the error spectra, for most of the above classes, are compact, as shown next.

Bus Order Error This class includes a common design error of incorrectly ordering the bits in a bus. For example, if the signals x_l and x_k of the bus with bits x_i, $i = 0, \ldots, n-1$ have been interchanged, the transform of a bus considered in isolation is:

$$AT(\tilde{f}) = \sum_{i=0}^{n-1} x_i * 2^i + x_k * 2^l - x_k * 2^k + x_l * 2^k - x_l * 2^l.$$

If the correct circuit transform was $AT(f)$, the transform of the faulty one is:

$$AT(\tilde{f}) = AT(f) + x_k * 2^l - x_k * 2^k + x_l * 2^k - x_l * 2^l.$$

The error polynomial has four nonzero spectral coefficients. In general, any permutation of bus signals will have the error transform with at most $2n$ spectral coefficients.

Bus Source Error This class replaces the intended source x_k with a source r_k. In this case, the AT of the error would be:

$$AT(e) = r_k * 2^l - x_k * 2^k.$$

Bus Driver Error This kind of errors corresponds to a bus being driven by two sources. It manifests itself in a way dependent on the implementation technology. For example, if the bus line is implementing "wired-OR", then by connecting an additional source r_k to a line x_k, the resulting signal is $x_k \vee r_k$. Using integer arithmetics, the logical OR is obtained as $x_k \vee r_k = x_k + r_k - x_k * r_k$. This identity leads to the following AT of the additive error:

$$AT(e) = (r_k - x_k * r_k) * 2^k.$$

Module Substitution Error A module is replaced by another module with the same number of inputs and outputs. Depending on the circuits replaced and their position in a logic network, various transforms can be obtained. In a simple example, where an AND gate producing $x_k = i_1 \wedge i_2$ is replaced by an OR gate producing $x_k = i_1 \vee i_2$, the error is transformed at the output as:

$$AT(e) = (i_1 + i_2) * 2^k,$$

because the logical AND transform results in $i_1 \wedge i_2 = i_1 * i_2$. By considering single gates, the *gate replacement* error model is included in this class.

Bus Single Stuck Line The error in which a bus is stuck at a constant value (0 or 1) is manifested as the additive error for which the arithmetic transform has n spectral coefficients. For this SLL error, the transform is:

$$AT(\tilde{f}) = \sum_{i=0}^{n-1} x_i * 2^i - \sum_{k=0}^{n-1} x_k * 2^k = AT(f) - \sum_{k=0}^{n-1} x_k * 2^k.$$

Hence the error transform equals

$$AT(e) = -\sum_{k=0}^{n-1} x_k * 2^k.$$

In the case of a bus stuck-at-1, the error transform has $2n$ nonzero coefficients:

$$AT(e) = \sum_{k=0}^{n-1} (2^k - x_k * 2^k).$$

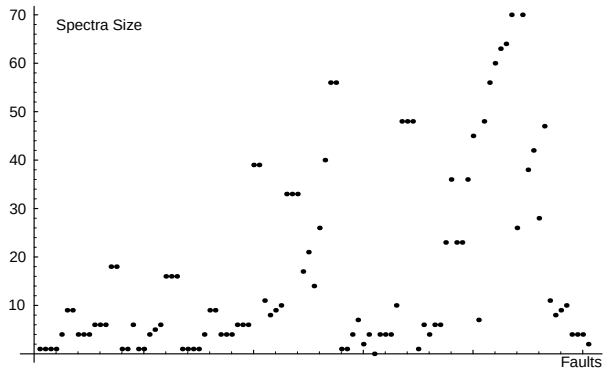

Figure 2. Spectrum Size Distribution of Stuck-at Faults in 4x4 Multiplier. X Axis: Faults in topological node order. Y Axis: Spectrum size

Any combination of the SSL errors would have the error transform that is linear in the number of lines that are stuck.

Single Stuck-At Faults In contrast to the above, the single stuck-at faults have a direct relation to testing of a circuit. These faults cannot be described by a single formula. For example, the distribution of spectra of all single stuck-at faults in an 4x4 CSA multiplier is plotted in Figure 2. This figure shows that a number of faults result in a substantial error spectrum. Regardless of the spectra size of the stuck-at faults, we demonstrate by the experiments that these faults are easily detectable by the vectors for small spectral error methods. The small additive error assumption is partly motivated by the examples of the common design error classes from [2]. There are several classes of circuits that are "small", such as the class of constant depth circuits. The results in [7] imply that this class of circuits has its (Fourier) spectrum small and concentrated in the low order coefficients.

4. Detecting Small Additive Errors

Under the assumption that the size of the additive error is bounded, the verification by error modeling can be done by using the bounds on the test vector size that are derived next.

The arithmetic transform of an arbitrary multi-output Boolean function can be obtained by multiplying the function values by the transform matrix T, defined in Equation 3. Rows of this matrix multiply the values that a function takes at all points of the function domain. A test set will contain a selected set of these points; finding out the polynomial representing a function can be performed by inverting the matrix. The structure of the matrix is identical to that of the RM error correcting code check matrix [8], albeit with a different addition operation. The redundancy incorporated in the matrix allows us to find a minimal test set for the case of a bounded spectrum error.

An error check matrix H_r consists of the T_n^{-1} rows corresponding to the points in the top $r+1$ layers of the lattice. This matrix has 2^n columns; the number of rows is equal to the number of points in these lattice layers. The auxiliary lemma, proven in Appendix, states that the matrix H_r has at least $2^{r+1}-1$ independent columns. The following theorem is used to derive a test set for the case of small spectra errors.

Theorem 1:*Any error that results in up to t spectral coefficients can be detected by examining $V = \sum_{i=0}^{\lceil log_2(t+1)\rceil -1} \left(\begin{array}{c} n \\ i \end{array} \right)$ points in $\lceil log_2(t+1)\rceil - 1$ upper layers of the lattice.*

Proof: By selecting all the points in the upper $\lceil log_2(t+1)\rceil - 1$ layers, we obtain a reduced matrix $H_{\lceil log_2(t+1)\rceil -1}$ of size $V \times 2^n$. It is sufficient to check that each $2t$ columns of this matrix are independent. Then, as in RM error correcting codes, any error polynomial with up to t terms will be detected. By the lemma in Appendix, the minimal number of independent rows is:

$$2^{\lceil log_2(t+1)\rceil -1+1} - 1 \geq 2t$$

Therefore, any polynomial with up to t terms will be uniquely identified. ■

This theorem uses the properties of the Arithmetic Transform that are identical to those of binary RM transform, due to the fact that both transforms consider binary inputs. A proof of the theorem for binary RM transform can be found in [8]. We note that a non-binary input generalization had been proven in [4] for detecting faults for circuits described by a multiple-valued RM transform. While the RM transform is of exponential size even for adders, the Arithmetic Transform is always of polynomial size, hence our result is more practical. Also, our result offers a generalization of the theorem in [8] for non-binary outputs.

This theorem provides an upper bound on the number of points that have to be simulated to detect this class of errors. In actual circuits, faults that involve many more spectral coefficients will be detected.

5. Experimental Results

We have performed a set of experiments over several most commonly used arithmetic circuits and several MCNC benchmarks. The errors considered include stuck-at faults and module replacement errors. Three types of gate replacement faults are applied. These faults substitute any gate in a circuit with AND, OR or XOR gate of the corresponding size. Faults are chosen to represent the basic design errors considered in Section 3.2. The stuck-at faults are selected for the additional reason of indicating the testing capabilities of this verification approach.

Circuit	single s-a-v			gate replacement		
	2nd	3rd	≤4	AND	OR	XOR
9symml	49.9	50.1	100	100	100	74.0
alu2	91.9	97.7	98.5	97.8	96.6	100
alu4	90.1	96.1	98.9	97.8	98.2	100
cordic	74.5	87.0	98.0	100	98.0	44.4
f51m	100	100	100	100	100	100
mux	90.0	100	100	100	100	100
my_adder	100	100	100	97.6	100	100
parity	96.7	96.7	100	100	100	N/A [1]
cm138a	67.5	77.5	97.5	100	100	100
decod	76.7	74.4	100	100	100	100
i1	86.9	86.9	86.9	50.0	66.7	85.7
z4ml	84.6	100	100	100	100	100

Table 2. MCNC Benchmark Coverage with lattice points up to 4 top layers

We recorded the coverage of these four classes of faults, as well as the statistics on the test set size. Tables 2 and 3 report the coverage of MCNC benchmarks and arithmetic circuits, respectively. The results indicate that only 4 layers are sufficient for the detection of the given classes of faults in arithmetic circuits, as well as most of the considered MCNC benchmarks. The goal of these experiments was to determine the effectiveness of the complete test set, including the information on the number of vectors per layer that detect each fault. Hence, no fault dropping was applied, and the reporting simulation time would not reflect the actual time needed in the verification. Otherwise, the running time is proportional to the circuit size and the number of vectors needed to detect the faults, which are reported.

It is worth noting that, compared with Theorem 1 and the experimental results considering the error spectra from Figure 2, a larger number of layers would have been expected. We interpret this by the fact that the theorem gives an upper bound on the number of vectors. Also these vectors are, according to Theorem 1, necessary for the unique identification of the error polynomial. We note that for the verification and testing purposes, only the detection of the presence of the error is required.

5.1. Improvements - Neighborhood Subspace Points

Although only up to 4 lattice layers (plus vector $11\ldots1$) were needed for good coverage, there were many redundancies in the test sets. This prompts us to consider the alternative schemes that use a subset of the considered lattice points.

We construct the vector windows covering exhaustively only the neighboring variables. We consider the neighboring inputs to be a_i, b_i, a_{i+1} and b_{i+1}. In adders, for ex-

[1] A purely XOR circuit - XOR gate replacement creates no errors

Circuit	Size	sav		AND		OR		XOR	
		lat	nei	lat	nei	lat	nei	lat	nei
Lookahead Adder	8x8	93.2	83.3	98.1	98.1	94.6	84.0	78.6	78.6
	12x12	94.3	83.8	98.8	98.0	95.2	85.5	81.1	79.8
	16x16	94.8	84.0	99.1	99.1	95.4	85.9	82.1	82.1
Ripple Adder	8x8	99.2	99.2	100	100	82.8	82.8	100	100
	12x12	99.5	99.5	100	100	83.7	83.7	100	100
	16x16	99.6	99.6	100	100	83.9	83.9	100	100
Array Multiplier	6x6	98.9	98.9	100	100	82.9	82.9	100	100
	7x7	99.1	99.1	100	100	83.7	83.7	100	100
	8x8	99.2	99.2	100	100	84.2	84.2	100	100
Booth Multiplier	6x6	94.2	92.1	92.3	91.3	98.9	97.3	91.0	89.9
	7x7	93.7	92.4	91.8	91.0	98.9	97.5	91.5	19.8
	8x8	94.5	92.8	92.4	91.8	99.1	97.0	91.6	97.5
ALU	8x8	99.7	76.8	100	71.4	100	78.6	96.6	83.1
	12x12	99.8	77.1	100	73.2	100	80.1	97.9	85.2

Table 3. Arithmetic Circuit Coverage: Lattice and neighbor points, 4 top layers

ample, the expression $c_i = (a_i + b_i)2^i$ joins together the neighbor variables in a polynomial term that is multiplied by the same constant. The size of the considered windows is in this case 4. Table 3 compares the two methods. First column for each fault type shows the coverage obtained by the exhaustive lattice points, and the second column reports the neighbor window results. We notice that little coverage is lost - further improvements are possible by experimenting with larger windows and alterinative schemes for neighbor signal selection.

Table 4 compares the test set lengths. Testing with all the vectors belonging to the top four layers requires $O(n^4)$ points. With the window size of four, the exhaustive test set for all the neighbor variable combinations needs only $O(n^2)$ points. The savings are equivalent to using two layers less in the test set.

This way of generating test vectors follows the universal test set approach, in which the tests are independent on the circuit implementation. Additional information used to restrict the test set came not from the circuit structure, but from the high-level specification.

6. Conclusions and Future Work

We proposed a vector-based verification of datapath circuits using the Arithmetic Transform and a concept of error modeling. We have shown that this approach can be applied to derive effective test sets for several classes of design errors. Furthermore, testing can be combined with the validation process through reusing of verification vectors for detecting manufacturing faults.

Through the arithmetic spectrum, the compact circuit representations and the capability of relating the common errors to the bounds on the vector set size are achieved. This provides the confidence in restricting otherwise exhaustive test set to its smaller subsets, without sacrificing the fault detection capability.

The improvements to the basic concept include the use of the high-level information on the input variable dependences, through neighbor window variables. More improvements are possible for circuits such as ALUs. Preliminary results show that dividers can be verified using the same approach. More work is needed in this direction. Further work on error modeling using this approach needs to be done.

References

[1] R. Bryant. On the Complexity of VLSI Implementations and Graph Representations of Boolean Functions with Applications to Integer Multiplication. *IEEE Transactions on Computers*, 40(2):205–213, February 1991.

[2] D. V. Campenhout, H. Al-Asaad, J. P. Hayes, T. Mudge, and R. B. Brown. High-Level Design Verification of Microprocessors via Error Modeling. *ACM Transactions on Design Automation of Electronic Systems*, 3(4):581–599, October 1998.

[3] Y.-A. Chen. Arithmetic Circuit Verification Based on Word-level Decision Diagrams. PhD thesis, Carnegie-Mellon University, School of Computer Science, May 1998.

[4] T. Damarla. Generalized Transforms for Multiple Valued Circuits and their Fault Detection. *IEEE Transactions on Computers*, 41(9):1101–1109, Sept. 1992.

[5] B. J. Falkowski. A Note on the Polynomial Form of Boolean Functions and Related Topics. *IEEE Transactions on Computers*, 48(8):860–864, August 1999.

[6] Y.-T. Lai, M. Pedram, and S. B. K. Vrudhula. Formal Verification Using Edge-Valued Binary Decision Diagrams. *IEEE Transactions on Computers*, 45(2):247–255, Feb. 1996.

[7] N. Linial, Y. Mansour, and N. Nisan. Constant Depth Circuits, Fourier Transform, and Learnability. *Journal of the ACM*, 40(3):607–620, July 1993.

Circuit	Size	sav		AND		OR		XOR	
		lat	nei	lat	nei	lat	nei	lat	nei
Lookahead	8x8	30	18	19	17	30	26	18	16
Adder	12x12	46	27	28	26	46	39	26	22
	16x16	60	36	45	35	56	52	42	30
Ripple	8x8	22	22	2	2	15	15	1	1
Adder	12x12	34	34	2	2	23	23	1	1
	16x16	46	46	2	2	31	31	1	1
Array	6x6	11	11	6	6	11	11	1	1
Multiplier	7x7	14	14	7	7	14	14	1	1
	8x8	15	15	8	8	15	15	1	1
Booth	6x6	31	22	17	15	21	18	18	18
Multiplier	7x7	7	28	18	16	23	19	24	19
	8x8	43	35	22	20	37	29	23	21
ALU	8x8	68	22	51	20	49	20	30	23
	12x12	74	25	65	28	58	24	35	32

Table 4. Arithmetic Circuit Vector Set Length Comparison

[8] R. M. Roth and G. M. Benedek. Interpolation and approximation of sparse multivariate polynomials over GF(2). *SIAM Journal of Computing*, 20(2):291–314, April 1991.

[9] C. Scholl, B. Becker, and T. M. Weis. Word-level Decision Diagrams, WLCDs and Division. In *International Conference on CAD*, pages 672–677. ACM, San Jose, California, November, 1998.

Appendix: Check Matrix H_r - Auxiliary Lemma

The check matrix H_r is obtained from the inverse transform matrix by selecting the evaluation points. The inverse of the AT transform matrix is defined as:

$$T_n^{-1} = \begin{bmatrix} T_{n-1}^{-1} & 0 \\ T_{n-1}^{-1} & T_{n-1}^{-1} \end{bmatrix} \quad (5)$$

(*Proof:*) By multiplying $TT^{-1} = I$.

Consider first an example with $n = 3$. If two top layers are taken, i.e. the points taken are 111, 011, 101 and 110. The check matrix H_1 is obtained from T_3^{-1}. Adding one more layer to H_1 results in a matrix:

$$H_2 = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The maximal number of independent columns of H_1, i.e. a *matrix rank*, is 4 by considering columns 4, 6, 7 and 8. Notice that not every four columns are independent, such as columns 5, 6, 7 and 8. However, any 3 columns are independent, and the minimum number of columns that are independent is 3. The structure of the matrix is such that this number can be found by inspecting the end columns, i.e. 5, 6 and 7. By taking more points, the rank is always equal to the number of rows, whereas the minimal number of independent columns is claimed to be $2^{r+1} - 1$, for H_r, with $r + 1$ top layers taken. This statement can be inspected by considering the $2^{r+1} - 1$ columns, corresponding to the points considered. Here, the minimal number of independent columns is 7. In general, the minimal number of independent columns increases every time a new layer is added, and is equal to the largest subspace contained in the tailing columns. The fact that the diagonal entries of the matrix are 1 ensures that the tailing columns are sufficient to consider, as the columns ahead cannot be linearly dependent to any tailing column. By inspecting H_1 and H_2, one can notice that the number of tailing columns considered is 2^{k+1}, for k layers, and that the first of these 2^{k+1} tailing columns is dependent on the rest. Hence, when a layer is added to H_k, a new 2^{k+1} vectors are added to the independent set of columns. We are now ready to formally prove the lemma, which is independent on the number of variables n.

Lemma 1: Matrix H_r has at least $2^{r+1} - 1$ independent columns.

Proof: By induction. Base step: we showed that for the case $k = 2$, the minimal number of independent column is $2^{k+1} - 1$.

In the induction step, for $k + 1$ layers, the minimal number of independent columns is equal to those for k layers, to which new 2^{k+1} columns are added. Then, the total number of independent columns is

$$2^{k+1} - 1 + 2^{k+1} = 2^{k+2} - 1 = 2^{(k+1)+1} - 1 \ \blacksquare$$